Worksheet 4 Programming language translators
Answers

Task 1

1. Fill in the blanks in the following:

   When a program is compiled from source code, the object code can be saved on disk and executed whenever required.

   An interpreter is useful for program development as parts of the program can be tested without having to recompile a lengthy program each time. However, a compiled program will generally run faster.

2. If you have written a program using a high-level language for one type of computer, can you run it on a different type of computer? Explain.

   Many high-level languages are portable because the source code can be compiled on different platforms (machine architectures). You would have to recompile the program into the appropriate object code using a specific compiler, or bytecode interpreter.

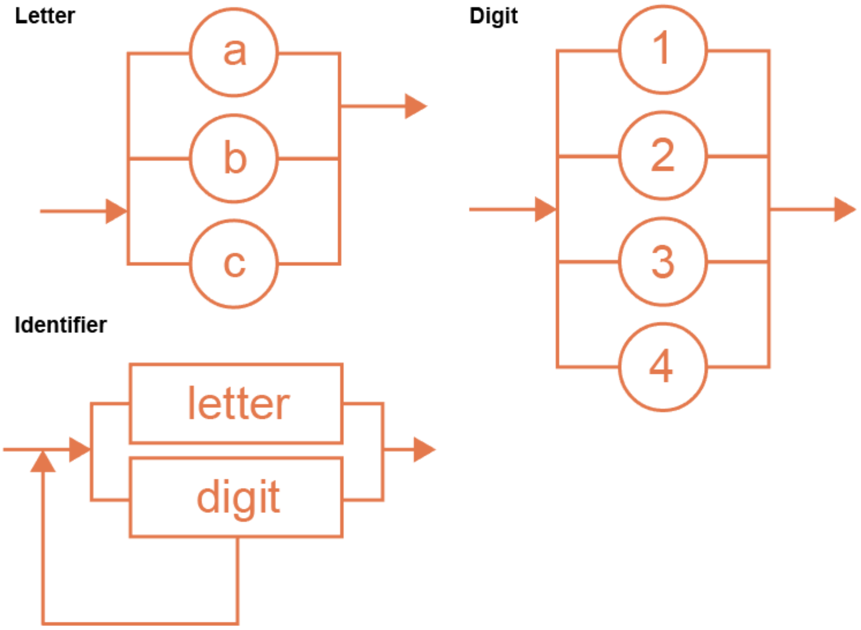
3. Converting code written in a high level language to bytecode might seem like a redundant process – why not just convert straight to machine code? What do you think the advantages of using a bytecode interpreter might be?

   - As long as a bytecode interpreter exists for a platform, the same code can be run on different platforms without having to worry about the hardware differences
   - There is a reduced risk for the user because they execute the bytecode interpreter (a trusted program) instead of directly executing downloaded compiled code which could contain a virus.

- It is possible, for example, to compile from Python into Java bytecode (using the Jython compiler) and then use the Java interpreter to interpret and execute it.

Task 2

Look at the following syntax diagrams which show some of the rules of a language. To read the diagrams, start on the left and follow the arrows.



Are the following sequences valid identifiers in this language?

1. a5 invalid, cannot have anything after a letter
2. d valid
3. abc invalid, cannot have more than one letter
4. 111 valid
5. 42c valid
6. 8b8 invalid, cannot have anything after a letter
7. 3B invalid, B is not an allowed symbol

Task 3

Compilers perform code optimisation to make the generated machine code more efficient. How might the compiler optimise the following code?

a)
`a = 5`
`b = (a + 240) * 0`
	`a = 5`
	`b = 0`
	Anything multiplied by 0 is 0, so the other part of the calculation is redundant and can be optimised out.

b)
`value = 0`
`WHILE value < 100:`
`    extra = 20`
`    value = value + extra`
	`value = 0`
	`extra = 20`
	`WHILE value < 100:`
	`    value = value + extra`
	`extra` is defined within the loop so it will be assigned every time the loop runs. Since its value doesn't change within the loop it can be defined once, before the loop.

c)
`temperature = 37`
`IF temperature < 100:`
`   PRINT "OK"`
`ELSE`
`   PRINT "Malfunction"`
	`temperature = 37`
	`PRINT "OK"`
	
	The value of the variable `temperature` is known at runtime and does not change. Therefore, as it is less than 100 the PRINT statement will be executed without the need for an IF statement.

d)
`numbers = []`
`VAT = 0.2`
`FOR i FROM 1 TO 1000:`
`   numbers[i] = i * (2 * VAT)`
	`numbers = []`
	`VAT = 0.2`
	`vat2 = VAT * 2`
	`FOR i FROM 1 TO 1000:`
	`   numbers[i] = i * vat2`
	The compiler may notice that calculating 2 * VAT (where VAT is a constant) does not need to be done 1000 times in the loop. It could create a variable vat2 to store this value once and then refer to it within the loop.